Software	— © PSS 1984
Manual Text	— © Orbis 1984
Software Authors	— D. Ritchie, T. Stoddard
Manual Author	— B.R. Morris
Typesetter	— S.Macdiarmid
Editor	— D.Cohen

Developed for Home Computer Advanced Course by Personal Software Services, 452 Stoney Stanton Rd, Coventry (0203) 81346.

Champ is designed to run on the Commodore 64, BBC Micro model B, and Sinclair Spectrum 48K.

It comprises an assembler for 6502/6510 or Z80 Assembly language, a program editor, and a monitor/debugger/disassembler. These facilities make Champ a powerful aid to the Assembly language programmer.

LOADING CHAMP

BBC Model B — CHAIN " "

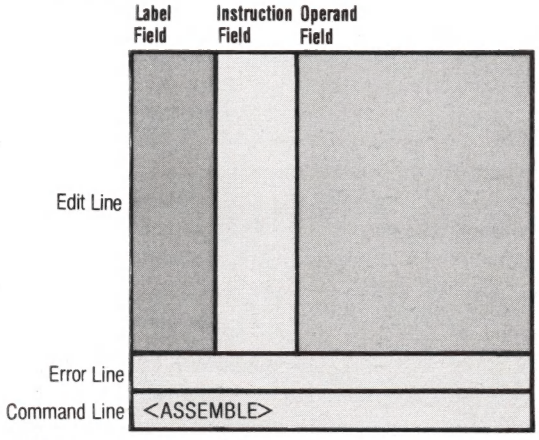
C64 — Hold down [SHIFT] and hit [RUN/STOP]

Spectrum 48K — LOAD " "

Champ will auto-run when loading is complete, so, having issued the LOAD command, you need do nothing until the screen clears and displays the copyright message. Stop the tape, remove it, and replace it with a blank data tape if you intend to save program files from Champ.

In addition to the copyright message on the screen, you will see a message about Champ's location in memory; this is important data, so make a note of it all now, even if you're not sure what it's for. When you've done that, hit [ESC] to run Champ.

The screen should look like this:



As you can see from the prompt on the Command Line, this is the <ASSEMBLE> mode; other modes are <EDIT>, <INSERT>, and <DEBUG>. The screen display in <INSERT> and <EDIT> modes is similar to that of <ASSEMBLE>, but the <DEBUG> screen is a different colour, and shows only the <DEBUG> prompt.

You use these modes for the following purposes:

<ASSEMBLE> mode
: is used after you have typed in an Assembly language program, in order to assemble it into machine code

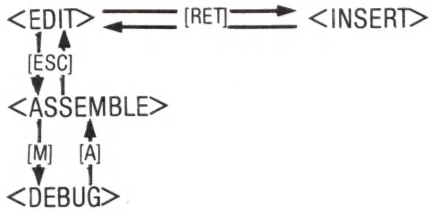
<INSERT> mode
: is what you use to type in an Assembly language program

<EDIT> mode
: enables you to modify an existing Assembly language program

<DEBUG> mode
: allows the inspection or modification of the contents of the memory, or the execution of a machine code program

Both <ASSEMBLE> and <DEBUG> modes are command modes. In these modes various keys represent commands which make something happen to your program or to memory. On the other hand, <INSERT> and <EDIT> are text modes; with these you can move program text around on the screen, and add to, or modify, it.

You can change from one mode to another as shown here:



If you have just loaded Champ, then there is no Assembly language program in memory, so <ASSEMBLE> mode cannot yet be used. Switch modes to <EDIT>, and thence to <INSERT>. You will see the prompt on the Command Line, and a blinking Cursor on the Edit Line.

Suppose that you wish to enter the following program:

```
DATA1   EQU  $23
STORE1  DB   $00
STORE2  DB   $00
          6502                        Z80
        LDA  #DATA1              LD   A,DATA1
        CLC                      AND  A
        ADC  #DATA1              ADC  A,DATA1
        STA  STORE1              LD   (STORE1),A
        ADC  #DATA1              ADC  A,DATA1
        STA  STORE2              LD   (STORE2),A
        RTS                      RET
;6502                    ;Z80
```

Type it in exactly as you see it; you will discover that Champ has automatic syntax and format checking. This means that you cannot exit from a line that contains a syntax error, or does not conform in layout with the three-field format of the screen. You will find that [CRSRL] and [CRSRR] move the cursor back and

KEY CONVENTIONS

A letter (or letters) enclosed in square brackets, thus, [A], means 'the key carrying this symbol'. Special keys referred to in this Manual are:

KEY	MEANING
[RET]	[Return key]; [Enter] on Spectrum
[ESC]	[Escape key]; [←] on C64; [Caps Shift + 1] on Spectrum
[CRSRR]	[Cursor Right]; [Caps Shift + 8] on Spectrum
[CRSRL]	[Cursor Left]; [Caps Shift + 5] on Spectrum
[↑]	[Cursor Up]; [Caps Shift + 7] on Spectrum
[↓]	[Cursor Down]; [Caps Shift + 6] on Spectrum
[CTRL]	[Control Key]; [Caps Shift] on Spectrum
[SP]	[Space Bar]

<EDIT> MODE COMMANDS

In <EDIT> mode, source text is displayed with the cursor on the Edit Line, and <EDIT> on the Command Line. Text on the Edit Line can be overwritten or deleted (using [DEL] or [SP]). [RET] causes the Edit Line contents to be checked for syntax and format. An error message will appear if the line is faulty, and the text will remain on the Edit Line. If the line is acceptable, it will be entered into the source text, and mode will change from <EDIT> to <INSERT>. [RET] toggles these two modes, while [ESC] toggles <EDIT> and <ASSEMBLE> modes.

The following keys can be used to move the source text on the screen, assuming the text on the Edit Line is correct. If a line is edited, and if the edited text is valid, then any of the following keys has the effect of entering the new line into the source text without changing the mode.

KEY	EFFECT
[↑]	Moves one line up the text.
[↓]	Moves one line down the text.
[CTRL]+[U]	Moves the screen text up one page.
[CTRL]+[D]	Moves the screen text down one page.
[CTRL]+[T]	Moves to the top of source text.
[CTRL]+[B]	Moves to the bottom of source text.
[CTRL]+[Z]	Deletes the Edit Line contents.
[ESC]	Enters <ASSEMBLE> mode.
[RET]	Enters <INSERT> mode.

N.B. The text movement keys have the same effects when used in <ASSEMBLE> mode, but they then do not require [CTRL] to be pressed. Thus [U] in <ASSEMBLE> mode moves the screen text up one page.

<INSERT> MODE COMMANDS

It is in this mode that you actually type your Assembly language program into the Assembler. The Command Line shows <INSERT>, and a flashing cursor appears on the Edit Line. The Edit Line (and the whole screen) is divided into three coloured columns, corresponding to the Label, Instruction, and Operand Fields of an Assembly language program:

Label Field
A label is any alphanumeric string of up to six characters. There must be a letter in the first position of the Field. A label does not require a colon (or any other character) as delimiter.

Instruction Field
Instructions are Assembly language mnemonics as in MOS Tech 6502 and Zilog Z80 specifications. They may be two, three, or four letters long, starting in the first position of the Field.

Operand Field
Operands may be hex constants (which must be preceded by $), labels, symbols, or expressions comprising two operands separated by + or –. Decimal, octal, and binary constants are not permitted. Operand formats for the various addressing modes are as specified by MOS Tech and Zilog.

Text entry in <INSERT> is subject to Field Formatting: This means it is impossible for you to type a seven-character label, or a five-character instruction. Typing an extra character, or hitting [SPACE], causes the cursor to skip to the first position of the next Field.

The [CRSRR], [CRSRL], and [DEL] keys act as normal in <INSERT> mode — subject to Field Formatting — but the delete key acts on the cursor character rather than on the character to the left of the cursor.

When you hit [RET] in <INSERT> mode, the contents of the Edit Line are checked for syntax and format; if an error is found, then a message appears on the Error Line. If no error is found, then the contents of the Edit Line enter the source text, and the Edit Line is cleared for the entry of a new line. Hitting [RET] when the Edit Line is blank toggles between <EDIT> mode and <INSERT> mode.

<DEBUG> MODE COMMANDS

This mode combines the following functions:
Memory Monitor — allows you to inspect and alter the contents of memory.
Hex Disassembler — allows you to interpret the contents of memory as machine code to be converted back into Assembly language.
Debugger — allows you to execute machine code programs in an error-trapping environment.

<DEBUG> is a command mode, but the Command Line/Edit Line/Field Format display of the other modes is not used: the screen is a blank page showing only the prompt and a cursor. In this mode all constants are hex constants without the '$' prefix, although the 'H' command supports decimal constants.

ABBREVIATIONS

addr	any hex address
saddr	start address of a block of memory
faddr	finish address of a block of memory (=1+ address of last byte of block)
daddr	destination address in hex
hx	a hex value (hx<=FF)
regname	CPU register name (see below)
expr	any arithmetic expression in one or two operands; operands may be decimal constants, '$' – prefixed hex constants, or legal symbols; operators are '+' or '–'
bystr	a string of hex byte values separated by spaces
chstr	a string of characters (exactly as it appears, no separators)

COMMAND	EFFECT
@ addr	Memory from the given address onwards is displayed one byte at a time, in hex and ASCII equivalent. Hit [RET] to advance to next byte, hit [ESC] to return to command level, or type a hex constant to replace the existing content of the byte
A	Return to <ASSEMBLE> mode
D addr	Memory from the given address onwards is displayed in screen pages; hit any key to continue, or [ESC] to return to command level
F saddr faddr hx	Every byte between saddr and faddr is filled with hx
M daddr saddr faddr	The block of memory between saddr and faddr is copied to the block starting at daddr
Q addr	Memory from addr onwards is disassembled; hit [RET] to continue, and [ESC] to return to command level
G addr	The code starting at addr is executed (returnable)
C addr	Execute from addr (non-returnable)
Bn=addr	A breakpoint number n, (between 1 and 8) is set at addr, to cause a break in execution of any program which accesses the contents of addr as an instruction; press [C] [RET] to continue from breakpoint
En	Eliminates breakpoint n
T	Displays the addresses of all the breakpoints
R regname	Displays the contents of a CPU register and accepts a new value (similar to the function of '@' above)
J addr	Executes the code from addr onwards, one instruction at a time, giving a full register display. Hit [J] to continue, [ESC] to return to the command level
H expr	Displays the decimal, hex, and binary value of expr
S bystr	Searches the memory from $0000 onwards for every occurrence of bystr. The word 'searching' is displayed while the program is searching, and the address is displayed when bystr is found. Hit [RET] to continue the search, or [ESC] to return to command level
Nchstr	As 'S' above
W	Load, Save, and Verify machine code to tape; see BASIC panel

CPU Regis Abbrevi

6502

A = Accumulator; X, Y = X, Y registers; P = Status register; SP = Stack Pointer

<ASSEMBL

COMMAND FORMATS

Find =>string [RET]
Searches the Assembly language program from the start of the program for the first occurrence of the given string.
Next =>string [RET]
Searches the Assembly language program for the next occurrence of the given string. The search begins from the end of the program line currently on the Edit Line.
Find =>[RET] and
Next => [RET]
As above, but this searches for the string defined in last 'F' or 'N' command. While a search is preceeding, the message 'searching' appears on the Error Line. If the search is successful, the line containing the string being searched for appears on the Edit Line. If the search is unsuccessful, the last line of the program appears on the Edit Line.
Load => Save => Verify =>
These must all be followed by a filename; double quotes are not needed, but the filename must be legal for the user's machine.
Print =>expression [RET]
This prints the hex value of the given expression on the Error Line. eg.
Print =>$F8-$C1 $37
Symbols already defined in source text can be used in expressions; but only one Operator (+ or –) is allowed per expression.
Quit =>[Y]
This quits Chasm, and returns control to the BASIC system only if [Y] follows the prompt;
any other response aborts the command.
[M]
Enter <DEBUG> mode. Return from there to <ASSEMBLE> mode by pressing [A] [RET].
[ESC]
Toggle <EDIT> and <ASSEMBLE> modes.
Assemble =>(option number) [RET]
This assembles the source text in one of a variety of ways, depending upon which numerical option is chosen:

ter Name ations

Z80

A = Accumulator; F = Flag/ Status register; H,L,B,C,D,E = H . . . E registers; SP = Stack Pointer; IX, IY = IX, IY registers.

> MODE

COMMANDS

KEY	PROMPT	FUNCTION
[F]	Find =>	Find a string
[N]	Next =>	Find a string
[L]	Load =>	Load a source file
[W]	Save =>	Save a source file
[V]	Verify =>	Verify a source file
[P]	Print =>	Print value of expression
[Q]	Quit =>	Quit to BASIC
[M]		Enter <DEBUG> mode
[ESC]		Enter <EDIT> mode
[A]	Assemble =>	Assemble program

To abort any of these commands, do not enter a command operand, just hit [RET] in response to the prompt.

Spectrum variations

KEY	PROMPT
[J]	Load =>
[S]	Save =>;
[sym.shift] + [R]	Verify =>

INSTRUCTION FORMATS

6502

INSTRUCTION		ADDRESSING MODE
LDA	#$D4	Immediate
LDA	$3C	Zero Page (Direct)
LDA	$A290	Absolute (Direct)
LDA	$31FE,X	Absolute Indexed
LDA	$7B,X	Zero Page Indexed
LDA	($2A,X)	Pre-Indexed (Indirect)
LDA	($2A),Y	Post-Indexed (Indirect)
CLC		Implied

Z80

INSTRUCTION		ADDRESSING MODE
LD	A,B	Register (Direct)
LD	A,$9F	Immediate
LD	($ED46),A	Absolute (Direct)
LD	A,(HL)	Register (Indirect)
LD	A,(IY+d)	Indexed (Indirect)
CCF		Implied

ASSEMBLY LANGUAGE FORMATS

PSEUDO-OP-CODES	MEANING
ORG addr	origin: assemble machine code in memory from addr onwards. The program line with ORG on it cannot take a label.
EQU	equate: set the symbol in the Label Field equal to the constant, symbol, or expression in the Operand Field.
DB const/ chstr	define byte(s): load this location, and as many following as required, with the value(s) of const or chrstr
DW const/ symb	define word: load this location with the lo-byte, and the next location with the hi-byte of the operand
DS const/ symb	define storage: add the value of the operand to the location address of this instruction

Abbreviations:

addr	a $-prefixed hex address
const	a $-prefixed hex constant; as an operand of DB, const must be a single-byte value. A string of constants, such as (DB const [SP] const [SP] const . . . etc) is valid
chstr	a string of characters enclosed in single quotes (e.g. 'AB3%9K10')
symb	any valid symbolic operand

ASSEMBLY OPTIONS

OPTION NUMBER	0	1	2	3	4	5	6	7
Display full list on screen	N	Y	N	Y	N	Y	N	Y
Load m/code into memory	N	N	Y	Y	N	N	Y	Y
Copy screen to printer	N	N	N	N	Y	Y	Y	Y
Verify labels, symbols & syntax	Y	Y	Y	Y	Y	Y	Y	Y
Display symbol table on screen	Y	Y	Y	Y	Y	Y	Y	Y

Y = This facility enabled
N = This facility disabled

E.g. **Assemble =>2 [RET]** causes the source text to be assembled with error-checking, and the resulting machine code to be loaded into memory as directed by the ORG pseudo-op-code. The symbol table is displayed on the screen, but no assembly listing appears on the screen, and there is no output to the printer.

Any option number can be preceded by 1, which gives a double-line display if the screen list facility is enabled.

If an error is found during assembly, a message will appear on the Error Line, assembly will cease, and the screen will display the source text with the faulty instruction appearing on the Edit Line.

LINKING MACHINE CODE AND BASIC

Once you're familiar with both Champ and Assembly language programming, you'll probably want to be able to call special-purpose machine-code routines from BASIC programs, rather than write entire programs in machine code. The easiest way of doing this is:

1) Using Champ, develop the Assembly language routine until it works.
2) From <ASSEMBLE> mode, SAVE the Assembly language routine to tape for future reference.
3) Assemble the routine into memory, choosing an ORG address near the top of User RAM (see your computer User Manual for Memory Map and advice).
4) From <DEBUG> mode, SAVE the block of memory containing your machine code to tape.
5) Quit Champ.
6) Write your BASIC program, starting with the instructions necessary to set the Top of User RAM pointers to an address safely below the ORG address of your routine. Follow those instructions in the program with a LOAD instruction that will load your machine code routine from tape to the location from which it was SAVEd (consult your User Manual).
7) Whenever you need to execute the machine-code routine in the BASIC program, use a CALL, SYS, or USR instruction with your routine's ORG address.
8) Save the BASIC program as usual.

If you exit from Champ to BASIC, and type LIST, you should see an example of this technique at work: When you LOAD Champ, you load only the short BASIC loader program which you see; when this is executed, it LOADS Champ itself as a machine code file into memory, then calls it as a machine code routine.

CHAMP ERROR MESSAGES

Error messages appear on the Error Line in all modes except <DEBUG>, which prints 'ERROR' at the current cursor position.

MESSAGE	MEANING
LABEL ERROR	A syntax or format error in the Label Field.
INSTRUCTION ERROR	A syntax or format error in the Instruction Field.
OPERAND ERROR	A syntax or format error in the Operand Field.
UNDEFINED LABEL	The Label or Symbol displayed on the Edit Line has not been assigned an address or a value.
JUMP OUT OF RANGE	The relative jump in the instruction on the Edit Line requires a displacement of more than 127 bytes forward or 128 bytes backward.
OVERFLOW	Assembling the instruction on the Edit Line into memory would overwrite CHAMP itself, or some protected memory, or would be out of range.
ERROR	The operand of a <DEBUG> command contains illegal symbols, or is too large a quantity, or is a bad address, etc.

forth on the Edit Line, and that [SPACE] sometimes produces space characters, and sometimes causes the cursor to skip from one field to the next. [DEL] erases the cursor character. Some of the effects of this Field-Formatted mode are strange at first, but you become used to them quickly. They should help you to produce error-free, neatly-formatted Assembly language programs.

When entering a new program, remember:

LABELS must start with a letter, and must not be more than six alphanumeric characters long.

INSTRUCTION MNEMONICS must be standard 6502 or Z80: two, three, or four letters long.

OPERANDS must follow standard 6502 or Z80 formats. They can contain arithmetic expressions comprising symbols or hex constants and a '+' or '—' operator, and can fill, but not exceed, the entire operand field.

COMMENTS must start on a new line with ';'. They can fill, but not exceed, the entire line, and are not subject to syntax or format checking.

When you have successfully typed in the program, enter <EDIT> mode. In this mode you can change the text on the Edit Line, and you can move the entire text file up and down on the screen using the following keys:

KEY	EFFECT
[↑]	Moves the Edit Line up one line
[↓]	Moves the Edit Line down one line
[CTRL]+[T]	Moves to the top of the text
[CTRL]+[B]	Moves to the bottom of the text
[CTRL]+[U]	Moves text up one screen page
[CTRL]+[D]	Moves text down one screen page
[CTRL]+[Z]	Deletes the contents of the Edit Line

These keys without [CTRL] have the same effects in <ASSEMBLE> mode, but you cannot delete or otherwise modify your text in that mode.

Switch to <ASSEMBLE> mode now, and hit [A] in order to assemble your program into machine code. The Command Line will show the Assemble => prompt. When you enter '11' [RET] you should see this double-line Assembly listing:

```
0000 00
0001 00
0002 A923
        LDA   #DATA1
0004 18
        CLC
0005 6923
        ADC   #DATA1
0007 8D0000
        STA   STORE1
000A 6923
        ADC   #DATA1
000C 8D0100
        STA   STORE2
000F 60
        RTS
;6502
DATA1 23    STORE1 0000    STORE2 0001
```

This may be a little puzzling at first, but it shows only the usual parts of an Assembly list: location address and machine code bytes on one line; label, instruction, and operand on the next line. At the bottom of the list is the Symbol Table, which lists all the program labels and symbols with their hex values.

Because you have not specified a start address in the program, you will see that the location addresses start at 0000. You must now return to <EDIT>, then to <INSERT> mode, and put a suitable ORG instruction at the start of the program. You must choose a location appropriate to your machine, and for this you need the information about Champ's memory usage given with the Champ copyright message. If you try to assemble code into an area which Champ protects, you will see the 'OVERFLOW' message on the Error Line during assembly. If that happens, you must change the ORG address.

When you've chosen an ORG address, and assembled your program without any error messages, assemble again, but this time use the '2' Assemble option so that the machine code is actually loaded into memory at its location address. Now hit [M], to switch to <DEBUG> mode.

The screen should now be all one colour, showing only the <DEBUG> prompt and cursor. Hit [Q] and type the ORG address of the Assembly language program, followed by [RET], and you should see the hex disassembly of your program. If you don't see this, you must check whether you have assembled it correctly, and whether the ORG address was correct, and whether it was really a RAM address. If you choose an ORG address in ROM, everything will seem to work, except that the machine code bytes will not be stored at their location addresses. You may need help from a memory map provided in your machine manual to check this.

If the disassembly is successful, then you can hit [G] followed by the ORG address, which will cause your program to be executed. Don't worry if you make mistakes with this, the worst that can happen is that you will cause a software crash, making it necessary for you to reload Champ and re-type the Assembly language. You can insure against this in part by doing a trial assembly to check your code, then SAVEing the Assembly language program on tape before trying to assemble it into its location addresses. This is similar to saving a BASIC program before trying to RUN it. Details of how to save source files are given overleaf.

If your program executes successfully, then the <DEBUG> prompt and cursor will return to the screen. The 'D' command can now be used to display the contents of the memory which the program should affect. If the results are successful, then you might want to SAVE the machine code (called the Object Code to distinguish it from the Assembly language Source Code) to tape, using the 'W' command in <DEBUG>. Having done that, you might like to try altering some of the object code in memory using the '@' command, also in <DEBUG>. Once you've started to understand roughly what's going on in Champ, you should simply play around with any and every command or option that meets your eye — you can't damage anything, and it's really the only way to learn.